LAPIN AMK
Lapland University of Applied Sciences

Creating New Features

Case: ARMA 3 SQF Syntax Scripting

Arto Alamäki

Bachelor's Thesis of the Degree Programme in Business Information Technology

Bachelor of Business Administration

TORNIO 2015

LAPIN AMK
Lapland University of Applied Sciences

Abstract of thesis

School of Business and Culture
Degree Programme in Business Information Technology

Author Arto Alamäki Year 2015
Supervisor Sari Mattinen
Title of thesis Creating New Features – Case: ARMA3 SQF Syntax Scripting
No. of pages + app. 42 + 22

Currently, the game ARMA 3 is lacking in features in some areas and to battle this, community members have created scripts to solve these problems. One of these problems is the lack of Active Protection System on tanks, which can be found in their real life counterparts. The script which tried to improve this situation, was the Bakerman Armour Improvement System. However, even this script is lacking in some basic features. This thesis seeks to extend the Bakerman Armour Improvement System script to its full potential by adding these missing features.

Similarities and discrepancies between the Armour Improvement System and the real life Active Protection System, in which the Armour Improvement System is based on, are examined in order to find a set of missing features. After finding a set of features, the SQF syntax scripting language, which is proprietary to ARMA 3, would be used to create these features. Furthermore, online publications regarding testing methods, user interface design, Active Protection System and game balance were used. Additionally, guides, forum posts and wiki pages created by ARMA 3 community members were also used. Constructive research method was used as the problem is practical and the solution for the problem would be practical as well.

During the feature design phase, it was found out that simply taking the features as they are in the real life Active Protection System and adding them to Armour Improvement System would possibly cause some balance issues in the game. To remedy this, some features were modified to have exploitable weaknesses to improve game balance. However, even these modifications remain uncertain as further testing would have to be conducted. Furthermore, the final script could be used as it is in present form, but future refinement and development is suggested.

Key words ARMA 3, Armour Protection System, Scripting, SQF syntax

CONTENTS

ABSTRACT

FIGURES AND TABLES

FIGURES AND TABLES

1 INTRODUCTION

This chapter discusses the background to the thesis topic and motivation in conjunction with the scope, objectives and the methodology used. The use of different sources and the structure of the thesis are also included and discussed.

1.1 Background and Motivation

Currently, the tactical military shooter game ARMA 3 is missing some features, which is why community members have started creating their own scripts to improve the situation. One of these problems is the lack of Active Protection System (hereinafter APS) found in real life tanks which is not replicated in the game. Therefore, an idea emerged to create a script for ARMA 3 that would mimic the functionality of an APS, specifically the Russian designed ‘Arena’ APS. However, ARMA 3 community member Bakerman had already created a version of this script and there was no need to create a second one, as their functionality would have been similar. In addition, the original script in question was released in September 2013 on Armaholic website. Since then the script has received minor updates regarding bugs. However, the original author, Bakerman, has dropped the development and support for the script in October 2014 (BI Military Games Forums 2014). This provided an opportunity to continue and expand on the original script by adding missing features.

Furthermore, my interest in games and programming acts as my motivation. Additionally, creating content for games gives me the opportunity to experiment with programming and further improve my skills, while providing endless opportunities to work on different platforms.

1.2 Scope and Objectives

This thesis focuses on designing and scripting new functions with a proprietary SQF scripting language for ARMA 3. Moreover, interface design theories, testing and game balance elements are discussed when new features for the APS are being designed and implemented. The research is also to include a discussion about different features of the

real APS platform Arena. These features and their characteristics are considered for replication in the game. Furthermore, the thesis is expected to yield practical results.

The research excludes hardware requirements for the game and tools used for creating the script. Additionally, the research is not an attempt to be a guidebook for its readers. The thesis should be partly treated as an experimental work and as an example of design and implementation of specific ideas. The testing phase was conducted offline to mainly test added features and their functionality in different cases. Testing excludes online and multiplayer scenarios.

Theories relating to interface design are considered and applied if the script reaches a certain level of complexity. If the script is simple and can be operated through a single drop down menu, which is already built into the game, there is no need for usability design. However, if the script requires its own menu, usability design is applied. Additionally, different software testing models are used in a form of black-box and unit testing.

1.3 Research Methodology and Limitations

Constructive research method is the most suitable for this kind of development work, as the problem is practical in nature and the solution is practical as well. Lukka (2000) has divided the constructive research process into 7 stages as depicted in figure 1 below.

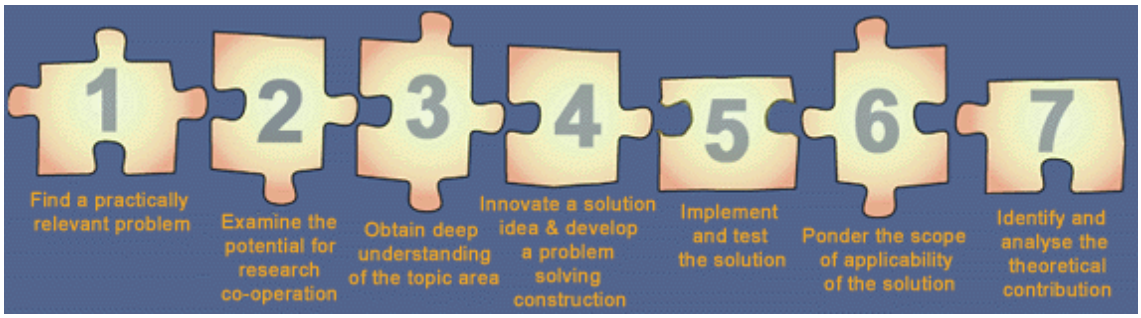

Figure 1. Typical research process (Lukka 2000)

However, this thesis research mainly concentrated on stages 1, 3, 4 and 5 as they are the most suitable for the purposes of this thesis. The first stage has already been discussed in previous chapters while the third stage was applied when finding information and

designing new solutions for the problems found. Stages 4 and 5 become apparent when new solutions are presented, implemented and tested within the scope of this thesis.

To achieve the objective of this developmental work, previous works and scripting documents created by users of the game ARMA 3 were analysed as main source of the information. Interviews and questionnaires are not necessary for achieving the final result.

This research assumes that the reader has a basic understanding of programming in order to grasp the code and design choices. Some information about the APS might be classified as a military secret, thus rendering some information un-obtainable for the researcher. Additionally, author's lack of experience with the SQF scripting language might limit the amount of features and complexity the final product has.

1.4 Regarding Sources

Currently the SQF is not widely documented scripting language and has very small amount of literature available. Because of this, sources used in this thesis refer to forums posts, wiki pages and community created guides. This is why most sources either have a web page name or authors are referred by their nicknames.

The script files created outside the original Bakerman Armor Improvement System (hereinafter Bakerman AIS) (Appendix 1 and 2) are produced by the thesis author and all modifications and additions to Bakerman AIS files are marked either as '//Added' or '//Modified' to clarify these changes. Additionally, the file 'defines.hpp' (Appendix 9) includes widely used and public styles for dialogs, which in this case were originally created by Iceman77 and have been used in this thesis. I take no credit nor do I claim any ownership of the code created by these two community authors.

1.5 Structure

The chosen platform and tools which were used are discussed and explained in chapter 2. Chapter 3 compares the real life APS to the original Bakerman AIS script and its current range of features. The discussion in chapter 3 leads to chapter 4 which proposes new

features based on previous chapters and discusses how these features could be implemented into the code. The possible game balance issues that these functions could produce are also included in chapter 4. Chapter 5 explains the design of these functions in detail. In chapter 6 and 7 every function is implemented and tested. Chapter 8 draws conclusion concerning the research.

2 PLATFORM AND TOOLS

This chapter discusses the platform used for the script and the tools which are being utilized to create various elements and assets for the script itself. The script created by Bakerman is also discussed briefly.

2.1 ARMA 3 and SQF

The platform for the script is the game ARMA 3, which belongs into a long lineage of military sandbox games created by Bohemia Interactive. The predecessor of ARMA series, i.e. Operation Flashpoint, came out in 2001 and has since seen many different iterations till the most modern one, i.e. ARMA 3, which came out in 2013 running on proprietary game engine Real Virtuality 4 (Bohemia Interactive Company Brochure 2014). Furthermore, the community is a very important part of the game series as it has provided a massive amount of extra content and improvements for the game.

As the engine and the SQF scripting language it uses are proprietary to Bohemia Interactive games, SQF cannot be compiled as the more traditional programming languages. Because of this, the game itself is needed to test the functionality of the script and can only be tested in this environment as assets and the script library are tied to this platform. The older SQS language has not used since the ARMA game series started (Bohemia Interactive Community Wiki 2014a). The game itself would be running on Windows 7 operating system.

2.2 Tools Used

SQF files can be created without any special tools. In Windows it can be done by creating text files and changing the file format from .txt to .sqf (Mikie J 2013). However, even when the Windows notepad could work as a script editing tool, it would be cumbersome when there is hundreds of lines of code. Therefore, the Notepad++ was chosen as it is a free source code editor is capable of editing SQF. By default, the SQF syntax highlighting is not supported in Notepad++, but ARMA 3 specific highlighting and auto completion plugin was installed to make the code more readable and easier to write. Additionally, the Gimp image editor and TexView 2 texture viewer were used for creating images suitable

for ARMA 3 environment. Microsoft Office 2013 was used for text editing and Draw.io website for creating a flow charts.

3 COMPARING APS AND AIS

This chapter discusses the real Arena APS and the Bakerman AIS script. Characteristics and features of these two systems are compared and taken note of when designing new features and modifying the script.

3.1 Arena APS and Bakerman AIS

APS can be divided into two different main categories, i.e. soft and a hard kill systems. A soft kill system mainly tries to guide the incoming projectile away from the target by disrupting its sensors or guiding systems. A hard kill system tries to destroy the incoming projectile before it hits the target with brute force, usually with high-explosives. Arena belongs to the latter category. These types of systems are usually attached to main battle tanks to enhance their survivability on the battlefield. (Meyer 1998, 7-9.)

The reason why Arena was chosen to be the source of information for the thesis is its maturity and the availability of data compared to other similar systems. However, Arena APS is still a military equipment and some information is expected to be concealed from the public. Moreover, a new version of the Arena APS was launched in 2013 under the title of Arena-3 (Defense Update 2013). Additionally, the features of Arena-3 are compared to the original Arena, but as the new system has not been well documented it was not used as a source of information for features.

The Bakerman AIS v0.1.3 is essentially an APS recreated with SQF scripts in ARMA 3. Originally, the script was supposed to be feature rich representation of the real APS, but Bakerman dropped the development for various reasons which are mentioned in his forum post regarding his script (BI Military Games Forums 2014). Moreover, this script provides the base for the added modifications and functions. All the information regarding the AIS was mainly collected by using the script and reading the code.

3.2 Projectile Detection

In order to detect incoming projectiles, APS's usually use a radar with varying coverage areas. In this case, the Arena APS uses a Doppler radar placed on top of the tank turret to scan for incoming projectiles. The radar itself covers 360 degrees, but the placement of the ammo cassettes limit it to 300 degrees while leaving a 60-degree blind spot at the rear of the tank turret. The radar will start tracing the projectile once it comes within 50 meters of the Arena APS and the system locks onto it at around 10 meters away from the tank. (Meyer 1998, 9.) However, since the new version of Arena, i.e. Arena-3, has been introduced, the coverage has increased to full 360 degrees (Defense Update 2013).

The AIS uses very similar approach in the script by scanning the direction of the incoming projectile via the function 'FNC_BAPS_SCAN' which is triggered by the 'fired' event handler shown in Appendix 1 (Bohemia Interactive Community Wiki 2014b). This function also includes the 'radar' coverage area divided into two different sectors, left and right, covering and arc of 340 degrees. If the incoming projectile is within one of these two sectors and heading towards the tank, it gets destroyed.

3.3 Controls and Reload Time

Functions and information has to be controlled by one of the tank operators. In this case, it is the tank commander, who has the access to the Arena control panel which controls the information received and sent to the system (Meyer 1998, 9). Currently, the control panel is missing completely from the AIS and no feedback is provided for the user. This type of panel is needed to control even the simplest features, like turning the APS on and off. Moreover, knowing the amount of ammo left and the direction of the incoming projectile are vital parts of the APS.

Currently, the reload time of 1.2 seconds in the AIS is longer than on the Arena APS. According to Meyer (1998, 9), after the first deflected projectile the Arena APS is ready to deflect the second incoming projectile within 0.2-0.4 seconds. This is considerably faster and could be changed to either represent the real time or made longer for better gameplay balance.

3.4 Physical Elements

As shown on the figure 2, the ammo cassettes are launched at an angle towards the incoming projectile and upon detonation, a shower of metal fragments fly towards the projectile incapacitating it. The explosion does not necessarily completely eliminate the threat, but rather reduces the lethality of it to safe level for the tank (Meyer 1998, 9). The fragmentation for the tank is not a problem, but for the surrounding infantry it is. When the cassette explodes, the fragmentation flies mainly towards the ground, but still covers a fairly large area to work as intended. As Meyer stated (1998, 9), the surrounding area up to 30 meters from the tank can be considered to be a dangerous sector for infantry units. Moreover, when the projectile gets destroyed, it too could possibly throw fragments beyond the 30-meter danger zone. This is the reason why turning the APS on and off is a very important feature.

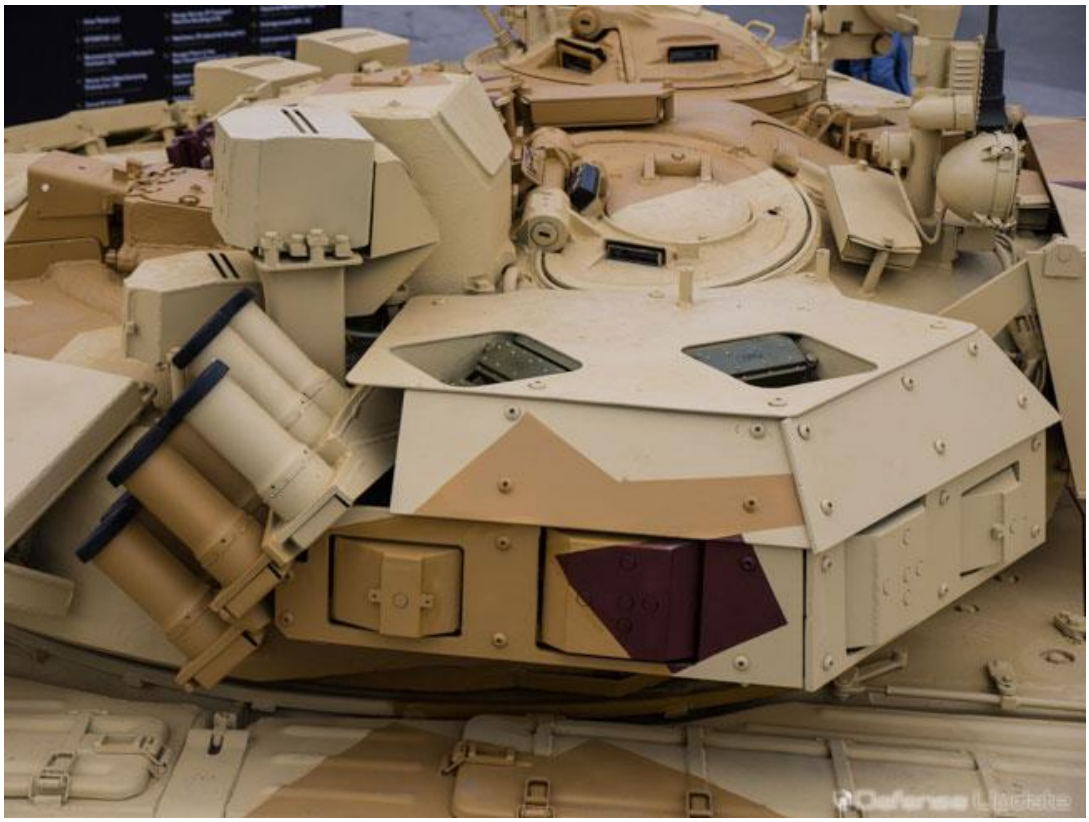

Figure 2. Arena-3 ammo cassette placement (Defense Update 2013)

Ammo cassette placements are modelled in some tanks in ARMA 3, but are missing in others. Despite this, the AIS is used in every tank as taking it away from others which do not have the model, could cause a huge difference in survivability and would favour tanks

which have it. Moreover, no physical representation has been used in the AIS script to mimic the launched ammo cassette as shown in figure 3. Additionally, other physical effects are partially missing from the script as only the explosion has been modelled. Adding the fragmentation effect would also bring the AIS closer to the APS and make it more realistic.

Figure 3. Arena ammo cassette (Fofanov 2003)

While adding physical objects to represent their real counterparts in the game would add realism, it would require 3D modelling, which unfortunately is not within the scope of this thesis and will not be implemented.

4 PROPOSED FEATURES AND BALANCE

This chapter goes through the proposed features and modification which are based on the differences and similarities between the Bakerman AIS and the real Arena APS. Additionally, the game balance issues these new feature additions could create are discussed.

4.1 Features and Modifications

Meanwhile the real APS has a specific panel that allows the tank commander to interact with the system, the graphical user interface (hereinafter GUI) is currently missing from the Bakerman AIS script. This GUI would house most of the visual cues for the APS and would provide important information for the commander. Such information as ammo count, direction of incoming projectile and reloading prompt could provide to be vital and enhance the survivability of the tank. Therefore, based on the previous chapters and discussions, the following functions and modifications are suggested as shown in table 1 below.

Table 1. Proposed features and modifications

Function/Modification	Purpose
Graphical user interface	Houses all functions
Radar	Warns about incoming projectiles
On/Off switch	Enables the activation and deactivation of the AIS
Modified defend angles	Provides game balance
Modified delay between defends	Provides game balance
Prompt ammo count	Displays the amount of ammo left in the AIS
Display reloading	Displays status text when the AIS is reloading

When comparing the features between the Arena APS and the Bakerman AIS, it is clear that the AIS lacks basic features, while still fulfilling the main feature of destroying incoming projectiles. By adding these proposed features, the AIS would become more usable and enhance the gaming experience. Additionally, defend angles and the delay between blocks could be modified to deal with some inherent balance issues discussed in the sub-chapter to follow.

4.2 AIS Possible Balance Issues

Inherently, armoured vehicles and especially tanks are stronger than any regular infantry unit in ARMA 3. Being stronger means that, one tank has the capability to dominate over one or several infantry units without the fear of getting destroyed. This could lead to frustration for other players and complaints about tanks being over powered could be made. However, to balance the situation, specialized anti-tank units (hereinafter AT-unit) carrying rocket launchers exist, but even so, they are not protected by thick steel armour as the tank is, but they do level the playing field to a point where the tank is not dominating anymore. However, if there are too many AT-units walking around, the life expectancy of a tank becomes short and, therefore, complaints about AT-units being over powered could be made. However, this does not necessarily mean that the game is entirely imbalanced, as there are many different variables to take into account. As Burgun (2011) describes, "A game being "balanced" is also always, at best, a rough approximation".

In this case, the balance between AT-units compared to the capabilities of the APS has been examined, as the APS could render the vehicle near impossible to destroy if it does not have any weaknesses or these weaknesses cannot be exploited with reasonable effort. For example, finding a different angle to attack from or to only fire a rocket at the tank when the APS is not active, are reasonably exploitable weaknesses in the system. Furthermore, where real systems are meant to have as few weaknesses as possible, some of these features could be handicapped on purpose in the script, to allow more level playing ground for all players. Moreover, these choices are argued for in chapters 5 and 6 where the separate features are designed and implemented. However, to truly find a good enough balance between units, extensive testing and especially online testing with players should be conducted, which, however, is outside of the scope of this thesis. Therefore, the balance is strictly based on what seems to be appropriate from the thesis author's point of view and testing would mainly concentrate on whether added functions work as intended or not.

5 DESIGNING FUNCTIONS

Based on proposed features and modifications, the following chapter discusses and argues for the design of each function. Additionally, what methodologies were used to reach the final structure of each function is also included.

5.1 Methods

As most features and functions are related to the GUI, guides and rules regarding interface design are applied. However, as the design of the GUI is very simplistic and houses only the essential means for displaying data and the necessary buttons to toggle the AIS on and off, no complex theories were used. Instead, the eight golden rules of interface design devised by Ben Shneiderman were adapted to an extent. This is because all rules might not be applicable in a small scale design. Especially the following rules were utilized: strive for consistency, offer informative feedback, prevent errors and permit easy reversal of actions (Shneiderman 2010).

5.2 Graphical User Interface

By default, ARMA 3 uses a list to display functions as illustrated in figure 4. This is not very intuitive and becomes extremely cumbersome when the user potentially has to scroll through a long list of commands and actions. Therefore, a separate GUI specifically for new features was created. This way cluttering the menu list was avoided and AIS specific functions were separated clearly from other functions in the game. However, to access the GUI, an action was attached to ARMA 3 default menu.

Figure 4. Default ARMA 3 GUI for actions

The GUI would hold the button for activating and deactivating the AIS and display the state of the AIS in text next to it. The ammo count for both sides of the tank would be held below the APS button. The amount of remaining ammo would be shown as numbers with letters L and R above them to indicate the left and right sides. Additionally, the right side of the GUI would be reserved for the radar function. To achieve better understanding of how the GUI would look like, the website Draw.io was used for creating a draft of the interface design which is illustrated in figure 5 below.

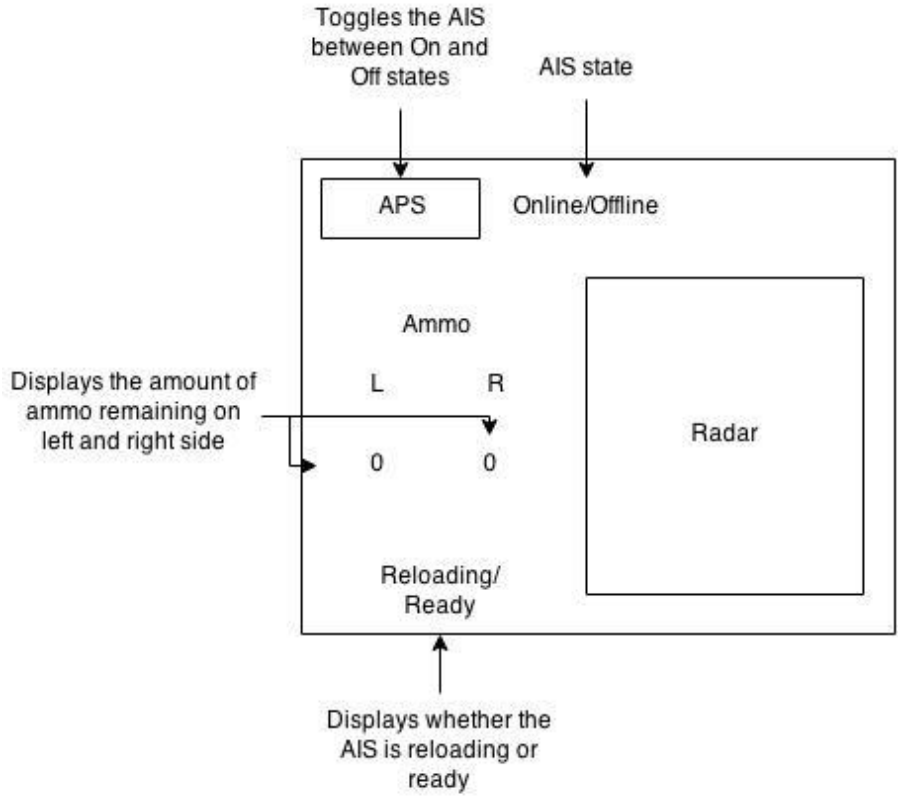

Figure 5. Draft design of the GUI

The colour scheme for the GUI was kept as simple as possible. Colours red and green were mostly used to indicate certain states of different features on the GUI. For instance, the colour of the sector on the radar would blink red when a projectile is approaching and the text prompting the reloading status is green when the system is ready to defend again.

5.3 AIS Activation

With this added function, the user could activate and deactivate the AIS at will. While having the AIS active all the time is the best solution for the tank crew, however, if infantry units are in close proximity of the tank when a projectile is being destroyed, it could possibly cause damage to the infantry (Meyer 1998, 9). Therefore, the deactivation features is a necessary addition. The activation function was attached to every spawned vehicle through the newly created GUI.

The activation of the real APS is controlled by the tank commander (Meyer 1998, 9). Therefore, the action was attached to the tank commander seat in ARMA 3. This improvement was made so that units outside the vehicle cannot activate and deactivate the AIS, but those who are inside the vehicle can, provided that the user is occupying the commander seat. Additionally, the idea behind of tying the function to the commander seat comes from divided responsibility inside the vehicle. A driver drives the tank, gunner aims and fires the cannon and the commander is in charge of electronic measures of the vehicle.

5.4 Defend Angles

Currently, the Bakerman AIS covers an arc of 340 degrees. From the gameplay point of view, this is only a good for the tank crew, as the tank crew would be covered from every angle and destroying such target could be near impossible task. However, the real Arena APS has an arc of 300 degrees which moves with the tank turret. Therefore, the tank has a weak point which could exploited by other players trying to destroy the tank. However, 300 degrees is still fairly wide arc considering that it moves as the turret moves. This still could prove almost impossible to get around and could cause frustration in other players and make most antitank weapons irrelevant. Therefore, the ‘dead zone’ in the AIS should be even bigger and the defence arc was lowered to 150 or 180 degrees. This leaves ample

room for exploitable weakness in the AIS. Moreover, using the AIS would not grant immunity anymore and the user of the APS should pay more attention to their surroundings to achieve maximum effectiveness of the system.

5.5 Modifying the Timed Delay between Blocks

Currently, the script already has a timed delay between defence situations. Whether it should be longer or shorter really depends on whether more realism or more balanced gameplay is wanted. If the time between defence situations is extremely short as it is in the real APS, the AIS could possibly defend multiple incoming projectiles at once, making it near impossible to destroy through conventional methods. This could make the vehicle carrying an AIS too overpowered and extremely un-balanced for the game. Thus, a longer delay between defences was introduced to counter the overpowering factor. With longer time between defences, a second projectile could be fired towards the target and the AIS would be incapable of reacting during set cool-off period. This opens an opportunity for the player to harm or possibly destroy the vehicle.

5.6 Radar

A missile radar can greatly contribute to the situational awareness of the tank crew. With the help of the radar system, the commander of the tank has the ability to tell the approximate direction of the incoming projectile. With this information, the commander has the ability to direct the gunner of the tank towards the general direction of the imposing threat. Moreover, as the Bakerman AIS script already knows the direction of the incoming projectile in a defence situation, that information should also be available for the commander. However, giving the exact direction of the projectile could render the radar too powerful as the commander could point out the exact direction and possibly the location of the player who fired it. The radar was implemented to have four sectors which light up if the projectile is coming from that approximate direction. This is illustrated in the figure 6.

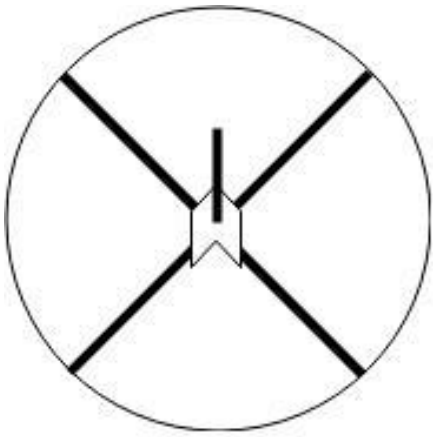

Figure 6. Draft of the radar

Moreover, in reality, different types of projectiles could be launched far away and then guided to the target, rendering the exact direction useless and misleading. Just by giving the approximate direction of the projectile i.e. front, right, rear or left would give sufficient information about the projectile source.

6 IMPLEMENTING FEATURES

This chapter discusses how each designed function was created and implemented into the Bakerman AIS script. The discussion about methods used to create these functions is also included.

6.1 Methods

In order to create to create the following functions, guides created by the ARMA 3 community members are referred to when constructing different functions for the script. Additionally, general good practises related to code formatting, comments and error prevention were used. Gary Willoughby (2006, 112) explains why and how these methods can help with legibility of the code and especially can make a difference from productivity point of view when the code grows and has to be re-visited later. Additionally, when the code was written, a small scale unit testing was conducted on each function to ensure integrity of the code (Fowler 2014). However, this is not documented as black-box integration testing would be conducted in a specific 'Testing' chapter of this thesis, i.e. chapter 7.

6.2 GUI Implementation

To create user interfaces in ARMA 3, Dialogs would be used. Dialogs can either be created manually by writing each interface class separately or with the GUI editor built in to the game (Bohemia Interactive Community Wiki 2014c). The latter was used in this thesis. This way, dialog classes could be created efficiently and adjusted if needed. Moreover, the GUI editor cuts down development time as there is no need to run multiple tests between minor changes in the class code. Once the editor has been accessed, the GUI elements could be placed down and their code copied and pasted into the dialog.hpp file as shown in Appendix 5. (Iceman77 2013.) As the code is now available and saved in a file, settings can be manipulated through different syntaxes used in scripts in the chapters to follow.

Figure 7 below displays the complete version of the interface. It should be noted that figure 7 was created after all scripts were fully in use, hence the text for APS activity, ammo count, reload and the radar indicators.

Figure 7. Completed GUI with scripts

The only element accessible and can be used by the user is the APS activation button. When pressed, it spawns the FNC_APS_ACTIVATION script, a function which will be expanded upon in the chapter to follow.

Furthermore, some settings were changed to achieve better usability from the GUI. Most of these setting were related to the background and text colours for better legibility. These changes would be in the defines.hpp file housing all styles for the GUI. As suggested by Iceman77 (2013), his version of the defines.hpp could be copied and used by community members as a base for control types (Appendix 9). Moreover, these styles are general and also displayed by the community wiki Dialog Control page (Bohemia Interactive Community Wiki 2014c).

By default, the GUI cannot be moved around from its initial spawning location on the screen and therefore, ends up obstructing the commanders' view. As instructed by the Dialog Control page (Bohemia Interactive Community Wiki 2014c), this was remedied by setting the Boolean variable of the 'movingenable' class control to 1. Additionally, a second class control 'moving=1' would have to be set on one of the GUI elements to allow the movement (Appendix 5).

In order to create the GUI, an ‘addAction’ syntax was set to spawn on every vehicle as shown in Appendix 2. The script in the file ‘call_GUI.sqf’ (Appendix 6) uses the ‘createDialog’ syntax for creating the interface set in the dialogs.hpp file. At the same time, the script checks whether the status of the AIS is online or offline and sets the status text in the GUI depending on the variable. The change in the text is made through the ‘ctrlSetText’ syntax which uses the dialog class ‘icd’ to identify which class would be manipulated (Iceman77 2014). Additionally, the FNC_UPDATE_GUI function is called to update the ammo count every time when the GUI is called to ensure that the ammo count is refreshed if the commander closes the GUI and a defence situation has happened during this time.

The reloading indicator uses the FNC_RELOAD_DISPLAY function (Appendix 8) which is spawned when a projectile has been deflected and the reloading process begins. The Bakerman AIS already has a script which degrades the performance of the AIS after every deflected projectile. The output of this script was used in the ‘sleep’ syntax which suspends the script for the given time. Additionally, the reloading function in itself has a 1.2-second delay. This has been taken in to account when the ‘sleep’ syntax has been used.

6.3 AIS Activation

Every spawned tank needs a variable which can be changed to indicate whether the status of the AIS is ON or OFF. The variable is changed through a script which retrieves the current status of the variable and changes it from 0 to 1 or vice versa as shown in Appendix 7. The code is called through the ‘APS’ button in the GUI. As previous stages have been completed, the variable is checked by the main Baked_AIS_fnc.sqf script when the AIS is active. If the variable in the vehicle is 0, then the APS is in OFF status and the incoming projectile will not be destroyed. The script was placed on its own separate .sqf file as it is its own distinct function that is being called. All vehicles are calling the same function, but only the variable in each specific vehicle, from which the function is being called, is changed. Figure 8 illustrates the toggle action of the function.

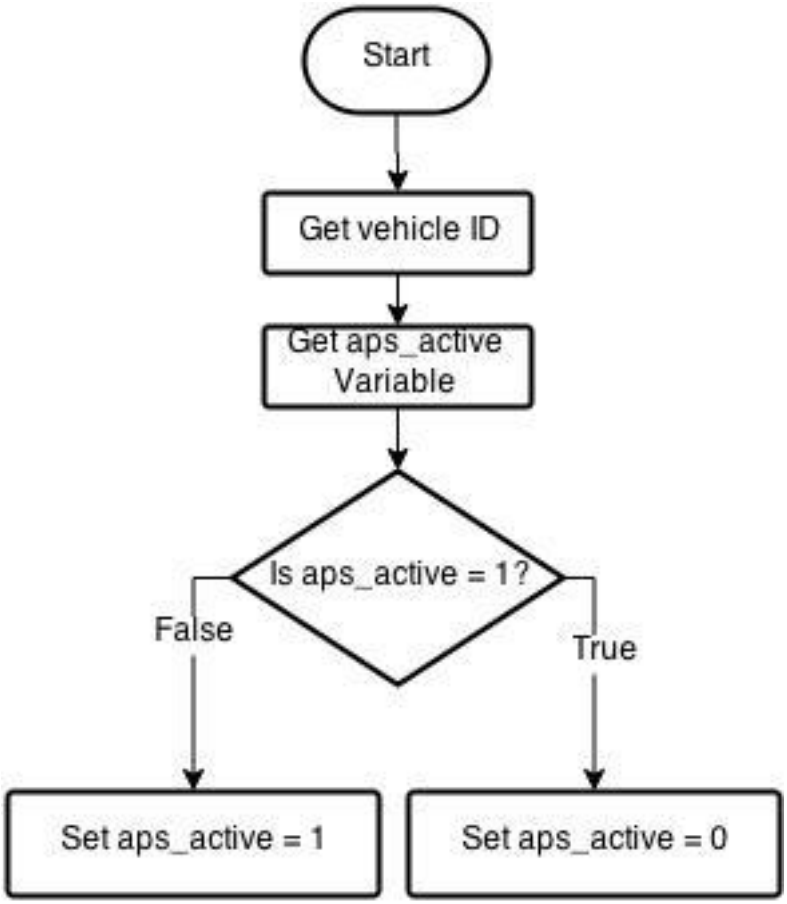

Figure 8. A flowchart of the added ON/OFF feature (modulated from Programiz 2014).

The ‘ONLINE’ message in the GUI for the AIS activation is tied to the vehicle variable. The variable is always checked when the GUI is spawned or the ‘APS’ button clicked. Additionally, the radar image is set to empty when the AIS is not active. Only when the AIS is active, the base image of the radar is displayed.

6.4 Ammo Count

In its current form, the AIS does not display any ammo count. This is extremely important information and in some cases vital. Currently, the crew of the vehicle can only guess how many projectile deflecting cassettes they have in their vehicle. With this information, the crew could choose their battles depending on their AIS ammo count or whether they should pull back and re-supply their AIS.

As selected vehicles in the ‘baps_defenders’ (Appendix 2) array spawn with a set ammo count, there is no need to create a separate variable. Instead, this variable was re-used by retrieving it with the script FNC_UPDATE_GUI as shown in Appendix 8. The function is called at the very end of the Baked_AIS_fnc.sqf to ensure that the GUI gets updated every time when a projectile gets deflected.

To display the current number of ammo cassettes left in the tank, the script changes the number on the GUI text boxes by ‘ctrlSetTex’ syntax and places the number retrieved from the vehicle into the ammo text field of the GUI as illustrated in figure 7.

6.5 Defend Angles

The defend angles are located in the existing function FNC_BAPS_SCAN (Appendix 1). The function retrieves the relative direction of the incoming projectile, which is in relation to the tank turret, and saves the value in to the ‘_relativeDir’ variable. The variable goes through two different direction check as shown in table 2.

Table 2. After changes to defend directions

```
if (_relativeDir > 0 && _relativeDir < 100) then //Modified
{
_block = "right";
};
if (_relativeDir > 260 && _relativeDir < 360) then //Modified
{
_block = "left";
};
```

If the value in ‘_relativeDir’ is within one of these checks, it determines on which side the blocking action happens. However, if the number does not fall into either of these two checks, the projectile continues its flight without obstructions from the AIS. Therefore, the values in these checks were changed to achieve the wanted sector angles. The angle was changed from 170 degrees to 100 degrees for the left and right sides.

6.6 Timed Delay between Blocks

As previously discussed the reload period exists in the script which uses ‘sleep’ syntax to pause the script to simulate reloading. Therefore, the time would have to be extended or made shorter depending on the wanted reload time. If realism was the main priority, the delay between defences would be lowered to 0.2-0.4 seconds, which would provide almost instant respond to incoming threats. However, the current reload time is set to 1.2 seconds, which is too short to be exploited reasonably. Therefore, the delay between defences was increased.

To find a suitable delay between defences, all AT-launchers were tested to find the time between subsequent missile launches. This test would exclude the ‘Titan MPRL Launcher’ as it is not capable of firing AT-rockets. The time was recorded with a stopwatch to get a rough estimate of how long it would take to each launcher to reload. The stopwatch would started when the launcher was fired and stopped when the launcher was reloaded and fired once again. This was repeated three times in a row in order to get an average time for each launcher. As shown in table 3, all launchers performed about the same.

Table 3. Average time between launches of 2 missiles.

Launcher	Average Reload Time (seconds)
PMCL	3.66
Titan MPRL Compact	3.55
RPG-42 Alamut	3.61
Average time of all launchers: 3.6 Seconds	

The time between defences was set to be two times longer than the average reload time of all launchers (7.2 seconds). It should be noted that the time is partly random and can be changed depending on what feels to be the best balance between good gameplay and usefulness of the AIS. The 7.2-second delay is suitable because, theoretically, during this time period one AT-unit could fire the rocket launcher twice. First shot being the one which gets destroyed and the second shot being capable of hitting the vehicle. Even when it is quite possible for one AT-unit to get a positive hit on the vehicle, using two AT-units would be recommended in order to maximise the effectiveness of this time window.

The delay was added to the ‘FNC_REALOAD_DISPLAY’ function. As shown in Appendix 8, the ‘sleep’ syntax was used to suspend the AIS script for 7.2 seconds. During this time, the AIS would not deflect any incoming projectiles, but the radar would still react and display the direction of the projectile.

6.7 Radar Visual Cue

The radar was divided into four 90-degree sectors. The middle point of these sectors would be facing north, east, south and west (Figure 9). Every sector would appear blank when the radar is in a non-active state. Depending on the direction of the threat, the corresponding sector would start to blink red to indicate incoming projectile. The direction of the projectile would be relative to the direction of the tank turret where the barrel would always be the 0-degree.

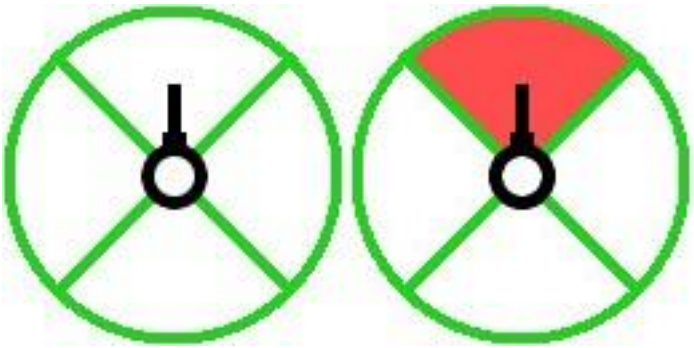

Figure 9. Example of the base image without any incoming threats (left) and an active image (right).

Images used on the radar were created with Gimp image editing software and saved in TARGA file format. After these images were converted to ARMA 3 appropriate file format (.paa) with TexView 2, they were stored with script files in the radar_img folder, which is accessed through scripted functions.

A separate ‘radar.sqf’ file was created to hold 2 functions FNC_DIRECTION_CHECK and FNC_RADAR which are shown in Appendix 3. The direction check function is called from the FNC_BAPS_SCAN once the relative direction of the incoming projectile is known. Once the function is called, the passed variable (_relativeDir) goes through the direction check where it is compared to four different 90 degree arcs. If the variable falls

into one of these arcs, the FNC_RADAR function is spawned and the '_radarDir' variable is carried over. The variable stored in _radarDir then determines which image is displayed in the GUI image box. The 'sleep' syntax is used set the rate between image changes to create a blinking effect on the radar. This is repeated up to 10 times before the script ends.

6.7.1 Audio Cue

This feature is active when an artificial intelligence unit or a player fires a projectile towards the target. Unlike the visual cue, the sound would be played until the projectile gets destroyed. The 'warning_sound.sqf' file was created to hold the code for the function (Appendix 4). The function is executed with 'execVM' syntax with passed variables '_projectile' and '_defender' from the function FNC_BAPS_BLOCK. These variables are used to identify the specific vehicle in which the warning sound is played. Additionally, the warning sound script runs check on whether the commander seat has been occupied by the player. If the seat is empty, the sound is not played. The sound was chosen from ARMA 3 sound library as this prevents the unnecessary creation of external sound files which could bloat the file size and possibly cause error if the file is corrupted or missing. Moreover, the sound 'alarm.wss' is already built in the game can be easily accessed through scripts (Hoffmann 2009, 56). Additionally, a while loop was created to loop the sound in 0.8-second intervals as long as the projectile is alive.

7 TESTING

Testing is a vital part of any software development and is needed to validate the integrity of the final product. Therefore, this chapter concentrates on testing newly added functions through different methods to find bugs and possible problems.

7.1 Black box testing

As previously discussed, small scale unit testing was done when the code was being written to ensure the code integrity. However, as the final testing concentrates on input and output, black box testing is most suitable (Software Testing Fundamentals 2015a). Moreover, instead of unit testing, every function were tested with integration testing, where functions interact with each other as a set (Software Testing Fundamentals 2015b).

Testing would be conducted in a single player environment in the game by creating scenarios suitable for added features. Feature specific test cases would be created for each function to validate the integrity of added features (Software Testing Fundamentals 2014). Original features created by Bakerman would be excluded from the test and only the modified parts would be tested.

7.2 Test Cases

As most added features are contained in the GUI, they could be tested simultaneously. Features which could be tested at the same time are AIS activation, ammo count, reload indicator and radar. However, the functionality of spawning the GUI would be tested first. Moreover, the following test case shown in table 4 would serve as a default template for all tests.

The GUI allows the interaction with all added functions in the AIS. The GUI itself can only be accessed through the tank commander seat. The table 4 below shows the test procedure for GUI creation

Table 4. GUI creation test case (modulated from Software Testing Fundamentals 2014)

Feature.	GUI creation
Prerequisites	Vehicle commander seat available/occupied by the player
Test Procedure	1.Player enters the commander seat 2. Executes GUI creation through ARMA 3 menu by scrolling and clicking on 'Tank Commander GUI' 3. Player clicks the frame and drags the GUI around the screen space
Expected Result	1. Commander GUI spawns on the middle of the screen and can be moved
Actual result	1. Commander GUI spawned and can be moved
Status	Success
Notes	Player camera cannot be moved when the GUI is on screen. GUI closed with 'Esc'
Date Tested	December 2, 2014

In order to deem the test successful, expected results would have to be achieved. In this case, the test was successful and expected results were met. However, when the GUI was created, a problem occurred where the player camera was stuck at one position and could not be moved until the GUI was closed.

The table 5 below is related to previous test case shown in table 4 as the GUI would have to be created before the AIS activation could be tested.

Table 5. AIS Activation test case (Modulated from Software Testing Fundamentals 2014)

Feature.	AIS Activation
Prerequisites	GUI must be open
Test Procedure	1. Player clicks the 'APS' button in the GUI 2. A missile is launched towards the vehicle
Expected Result	1. The text next to the button changes as the button is clicked 2. When 'Online' projectile gets destroyed and radar is displayed 3. When 'Offline' projectile goes through and radar position appears blank
Actual result	1. Text changes according to status 2. While 'Online', projectile gets destroyed and radar is displayed 3. While 'Offline', projectile goes through and radar position appears blank
Status	Success
Notes	
Date Tested	December 2, 2014

When the AIS activation button was clicked, the status text and the radar image changed according to AIS status. When projectiles were launched towards the tank while the AIS was offline, the tank was destroyed. While the AIS was online, projectiles were deflected accordingly. The test for the AIS activation was successful without any complications or errors.

The test shown in table 6 required a special environment to be created for the radar function. Like previous test cases, the GUI would have to be created in order the see the radar image.

Table 6. Radar test case (Modulated from Software Testing Fundamentals 2014)

Feature.	Radar
Prerequisites	GUI must be open
Test Procedure	1. A projectile is launched from every direction at least once.
Expected Result	1. Corresponding sector to the projectile on the radar starts to blink when the projectile is approaching 2. Warning sound plays while the projectile is approaching and alive
Actual result	1. Sectors blink according to the projectile direction 2. Warning sound plays while projectile alive and approaching
Status	Success
Notes	1. The warning sound is muffled due to loud tank engine
Date Tested	December 3, 2014

The test included multiple different scenarios where AI units fired rocket launcher towards the tank while the AIS was active. In one scenario, four AI units were placed left, right, front and rear of the tank to test all sectors. Every sector responded accordingly to incoming projectiles in this scenario. Other similar scenarios used also yielded similar results and therefore, the test could be concluded as a success. However, when the engine in the tank was turned on, the sound of the engine was fairly loud which made the warning sound to appear quiet.

The ammo count test in table 7 was conducted in conjunction with the radar test as it required an open GUI and rockets being launched towards the tank. Additionally, the AIS had to be in active state.

Table 7. Ammo count test case (Modulated from Software Testing Fundamentals 2014)

Feature.	Ammo count
Prerequisites	GUI must be open
Test Procedure	1. A projectile launched towards the tank with the AIS active
Expected Result	1. Ammo count goes down on the corresponding side after every deflected projectile 2. Closing and re-opening the GUI does not reset the ammo count
Actual result	1. Ammo count goes down on the corresponding side after every deflected projectile. 2. Closing and re-opening the GUI does not reset the ammo count
Status	Success
Notes	
Date Tested	December 3, 2014

The difference, however was the sector in which these projectiles were fired from. Every projectile had to fall into the AIS defence sector where the projectile would be deflected, otherwise the reload script would not be activated and the ammo counter would not reduce 1 ammo from the AIS. Additionally, between deflected projectiles, the GUI was closed and re-opened to ensure that the ammo count would not reset. During the test, ammo counter worked as intended and the counter did not reset itself after re-opening the GUI.

After the AIS deflects a projectile, the reload prompt function is activated and the text in the GUI changes according to the current status. Table 8 illustrates the test run for the reloading prompt function.

Table 8. Reloading prompt test case (Modulated from Software Testing Fundamentals 2014)

Feature.	AIS reloading prompt
Prerequisites	GUI must be open and the AIS should have ammo left
Test Procedure	1. A projectile launched towards the tank with the AIS active
Expected Result	1. The prompt for reloading displays either ‘Reloading’ or ‘Ready’ depending whether a projectile has been deflected or not 2. ‘Reloading’ text stays on while the AIS is inactive and reloading
Actual result	1. The prompt for reloading displayed ‘Reloading’ and ‘Ready’ depending whether a projectile was deflected or not 2. ‘Reloading’ text stays on while the AIS is inactive and reloading
Status	Success
Notes	
Date Tested	December 3, 2014

In a defence situation, the reloading prompt changed from ‘Ready’ to ‘Reloading’ and remained on until the AIS was ready to defend again. Like in the previous test, the GUI was closed and re-opened during the test to find out whether reloading prompt would reset itself to its default state and display wrong information. However, this did not happen and the test was successful and yielded expected results.

When the AIS defence arch was changed, the new setting was tested by firing projectiles at the tank within and outside the new sector as illustrated in table 9. Unlike previous tests, the creation of the GUI was not required.

Table 9. Modified defend angles test case (Modulated from Software Testing Fundamentals 2014)

Feature.	Modified defend angles
Prerequisites	Tank must be alive and AIS active
Test Procedure	1. A projectile is launched towards the tank from the -100 and +100 degrees sector from the 0 degree point (tank turret barrel) 2. A projectile is launched towards the tank from the blind spot between 100 and 260 degrees
Expected Result	1. The projectile should be deflected by the AIS when it is within the sector 2. The tank should be vulnerable from this angle and projectile should go through and damage the tank
Actual result	1. Projectile was deflected successfully 2. Projectile went through and hit the tank
Status	Success
Notes	
Date Tested	December 3, 2014

In order to test this function, the tank had to be alive and in operational condition. Moreover, the AIS had to be active, otherwise, every launched projectile would go through the AIS and damage the tank. While the AIS was active, projectiles inside the AIS defence sectors, left and right, would be deflected and projectiles outside these sectors would go through and hit the tank. All sectors worked as intended and the test could be deemed successful.

The newly added delay feature was a rough approximation of what could be good and exploitable weakness in the AIS. Table 10 below shows the process taken to test the new added delay between defences.

Table 10. Delay between defences test case (Modulated from Software Testing Fundamentals 2014)

Feature.	Delay between defences
Prerequisites	Tanks must be alive and AIS active
Test Procedure	1. Second projectile is launched towards the tank right after the first projectile has been deflected by the AIS 2. Second projectile is launched towards the tank after waiting at least 8 seconds after the first projectile has been deflected
Expected Result	1. The projectile goes through while the AIS is reloading 2. The projectile is destroyed when the AIS has done reloading
Actual result	1. The projectile went through while the AIS was reloading 2. The projectile was destroyed once the AIS had reloaded after 8 seconds
Status	Success
Notes	
Date Tested	December 3, 2014

As the delay would start after one projectile has been deflected by the AIS, a secondary projectile was launched after the first, in order exploit the time gap in the reload script. In this case, the first projectile was deflected, the second projectile was not. Additionally, a third projectile was launched after waiting for a while to confirm that the AIS had reloaded itself and was ready to deflect again. As expected, the third projectile was deflected and the test was successful.

All presented test cases through tables 4 to 10 ended up being successes. This is due to initial testing done during the scripting phase where most functions were tested individually before moving to black box testing. However, this does not mean that all functions worked completely without problems, as can be seen on tables 4 and 6 where some problems were discovered in the notes. Moreover, as these tests were short term and only run in a single player environment, possible bugs and problems could appear in multiplayer testing, which is not included in the scope of this thesis.

8 CONCLUSION

The aim of this thesis was to create new added features to an existing script while using a real life example of the system as a template. The Arena APS proved to be a good source of information for added features. However, if these features were to be replicated to one to one scale, they would have caused major issues with game balance and thus were adjusted to have weaknesses to improve balance. This adjustment taught me that directly transforming features from real life counterparts to a game could be more troublesome than expected and not as a straightforward process as imagined.

Writing the code for each function was a fairly simple process once the correct syntaxes were found. Moreover, having previous experience in programming was useful as the SQF uses similar syntaxes to other programming languages. The script could be used in its current form, but serves as a platform for future development as well. However, while the script works as intended, further development and online testing should be conducted if the script were to be released for wider public distribution.

The most challenging stage of the thesis was to find relevant material regarding the SQF scripting language. The material is extremely scarce and limited to a few community created guides which are valuable for giving a good basic knowledge about the language and its capabilities. However, to get beyond what these guides offer, users have to rely on community forums and to dissecting existing scripts for information. It would be recommend that users interested in SQF scripting would familiarize themselves with these guides before starting their own scripting work. Additionally, reading through scripts created by other community members can notably facilitate the process.

REFERENCES

BI Military Games Forums 2014. Armour Improvement System (AIS). Referenced November 31, 2014.
http://forums.bistudio.com/showthread.php?161726-Armour-Improvement-System-%28AIS%29&p=2790129&viewfull=1#post2790129

Bohemia Interactive Company Brochure 2014. Downloaded January 12, 2015.
http://www.bistudio.com/assets/pdf/bohemia_interactive_brochure_2014.pdf?2

Bohemia Interactive Community Wiki 2014a. SQF syntax. Referenced November 31, 2014.
https://community.bistudio.com/wiki/SQF_syntax

Bohemia Interactive Community Wiki 2014b. Arma 3: Event Handlers. Referenced December 1, 2014.
https://community.bistudio.com/wiki/Arma_3:_Event_Handlers

Bohemia Interactive Community Wiki 2014c. Dialog Control. Referenced December 2, 2014.
https://community.bistudio.com/wiki/Dialog_Control

Burgun, Keith 2011. Understanding Balance in Video Games. Referenced October 31, 2014.
http://www.gamasutra.com/view/feature/134768/understanding_balance_in_video_.php

Defense Update 2013. New Arena-3 APS Debut At Rae-2013. Referenced November 13, 2014.
http://defense-update.com/20130925_a-new-arena-aps-debut-at-rae-2013.html#.VHsbT8naeTt

Fofanov, Vasiliy 2003. ARENA Active Protection System. Referenced November 31, 2014.
http://fofanov.armour.kiev.ua/Tanks/EQP/arena_elem.jpg

Fowler, Martin 2014. UnitTest. Referenced January 10, 2015.
http://martinfowler.com/bliki/UnitTest.html

Hoffmann, Sascha 2009. Armed Assault Editing Guide - Deluxe Edition. Downloaded November 3, 2014.
http://www.armaholic.com/page.php?id=4847

Iceman77 2013. Dialog Tutorial For Noobs V3. Downloaded November 17, 2014.
http://www.armaholic.com/page.php?id=18363

Lukka, Kari 2000. The Constructive Research Approach. Referenced November 31, 2014.
http://www.metodix.com/en/sisallys/01_menetelmat/02_metodiartikkelit/lukka_const_research_app/kooste

Meyer, Tom J. 1998. Active Protection Systems. Impregnable Armour or Simply Enhanced Survivability, ARMOR Iss. May-June. Downloaded November 1, 2014. http://www.benning.army.mil/armour/eARMOR/content/issues/1998/MAY_JUN/ArmourMayJune1998web.pdf

Mikie J 2013. Fockers ARMA 3 Scripting Guide. Downloaded January 28, 2014. http://www.armaholic.com/page.php?id=20465

Programiz 2014. Flowchart In Programming. Referenced November 31, 2014. http://www.programiz.com/article/flowchart-programming

Shneiderman, Ben 2010. The Eight Golden Rule of Interface Design. Referenced November 20, 2014. https://www.cs.umd.edu/users/ben/goldenrules.html

Software Testing Fundamentals 2014. Test Case. Referenced December 2, 2014. http://softwaretestingfundamentals.com/test-case/

Software Testing Fundamentals 2015a. Black Box Testing. Referenced January 3 2015. http://softwaretestingfundamentals.com/black-box-testing/

Software Testing Fundamentals 2015b. Integration Testing. Referenced January 3 2015 http://softwaretestingfundamentals.com/integration-testing/

Willoughby, Gary 2006. Purebasic A Beginner's Guide to Computer Programming. Downloaded 11 January 2015. http://www.purearea.net/pb/download/PureBasicBook.pdf.

APPENDICES

Appendix 1	Bake_AIS_fnc.sqf
Appendix 2	Baked_AIS_init.sqf
Appendix 3	Radar.sqf
Appendix 4	Warning_sound.sqf
Appendix 5	Dialogs.hpp
Appendix 6	Call_GUI.sqf
Appendix 7	Aps_Activation.sqf
Appendix 8	Update_GUI.sqf
Appendix 9	Defines.hpp

```
// FILE. Baked_AIS_fnc.sqf
// AUTHOR. Bakerman, Extended/modified by Arto Alamäki
// LAST EDIT. 15/09/2013
// Version. 0.1.3
// DESCRIPTION. APS functions for Baked_AIS (Armour Improvement System)
// LICENSE. GO CRAZY - USE IT - ABUSE IT - CREDIT IF YOU WISH

// Called on "fired" event
FNC_BAPS_FIRED =
{
private ["_unit","_weapon","_projectile"];
_unit = _this Select 0;
_weapon = _this Select 1;
_projectile = _this select 6;
// Check if weapon used is a launcher
for "_i" from 0 to ((count baps_weapons) - 1) do
{
if (_weapon == (baps_weapons select _i)) then
{
[_projectile, _unit] call FNC_BAPS_SCAN;
};
};
};

// Projectile scanning for targets
FNC_BAPS_SCAN =
{
private
["_projectile","_unit","_vehicleType","_defender","_block","_i","_weaponDir","_weap
onDeg","_vehicleDir","_turretDir","_relativeDir","_dir","_fov","_dis"];

_projectile = _this Select 0;
_unit = _this Select 1;
for "_i" from 0 to ((count baps_vehiclelist) - 1) do
{
_defender = baps_vehiclelist select _i;
if (!alive _defender) exitWith  {};
if (count (crew _defender) == 0) exitWith {};
// By default any vehicle with APS may block
_block = "true";
// Specific duel APS code for Merkava Mk4
_vehicleType = typeOf _defender;
if (_vehicleType == "B_MBT_01_cannon_F") then
{
// Get the direction of the tank's turret
_weaponDir = _defender weaponDirection "cannon_120mm";
_weaponDeg = (_weaponDir select 0) atan2 (_weaponDir select 1);
```

```
if (_weaponDeg < 0) then { _weaponDeg = _weaponDeg + 360; };
_vehicleDir = direction _defender;

_turretDir = _weaponDeg - _vehicleDir;
if (_turretDir < 0) then { _turretDir = _turretDir + 360; };
_relativeDir = [_defender, _projectile] call BIS_fnc_relativeDirTo;
_relativeDir = _relativeDir - _turretDir;
if (_relativeDir < 0) then { _relativeDir = _relativeDir + 360; };
// Remove the default blocking
_block = "false";
// Set blocking direction and angle limitations
if (_relativeDir > 0 && _relativeDir < 100) then //Modified
{
_block = "right";
};

if (_relativeDir > 260 && _relativeDir < 360) then //Modified
{
_block = "left";
};
};
[_relativeDir]call FNC_DIRECTION_CHECK;//Added
if (_block != "false") then
{
_count = 0;
// Set the field of view of the projectile scanner
_fov = 1;
while {_count < 10} do
{
// Scan for vehicles in-front of the projectile
_dir = [_projectile, _defender] call BIS_fnc_relativeDirTo;
if (_dir > 330) then
{
_dir = 360 - _dir;
if (_dir < 0) then { _dir = _dir + 360; };
};
_dis = _defender distance _projectile;
if (_dis < 50) then { _fov = 20 };
if (_dis < 25) then { _fov = 30 };
if (_dir > -_fov && _dir < _fov) then
{
[_defender, _projectile, _unit, _block] spawn FNC_BAPS_BLOCK;
_count = 11;
} else
{
_fov = _fov + 1;
};
_count = _count + 1;
```

```
sleep 0.03;
}
};
};
};

// Threat elimination functions for vehicles
FNC_BAPS_BLOCK =
{
private
["_defender","_projectile","_unit","_block","_canBlock","_canBlockLeft","_canBlockR
ight","_speed","_time","_defendDistance","_wait","_sleep","_reloading","_listed","_am
mo","_ammoLeft","_ammoRight", "_reloadLeft", "_reloadRight","_active"];
_defender = _this Select 0;
_projectile = _this Select 1;
_unit = _this Select 2;
_block = _this Select 3;
//Checks if the APS is on or off
//Added
_active = _defender getVariable "aps_active";
if (_active == 0) exitWith {};
[_projectile, _defender] execVM "warning_sound.sqf";//added
// By default the vehicle can block from all directions
_canBlock = true;
_canBlockLeft = true;
_canBlockRight = true;
// Get defender damage
_damage = getDammage _defender + 1;
_do = 1;
while {_do == 1} do
{
_speed = speed _projectile;
if (_speed != 0 && alive _projectile) then
{
// Calculate the flight time of the projectile
_dis = _defender distance _projectile;
_time = _dis / 1000 / _speed * 60 * 60;
// Adjust defend distance based on speed
_speedAdjustment = _speed / 110;
// Constant minimum distance to defend
_defendConstant = 10;
// Degrade performance for damage
_degradeDamage = _damage * _damage;
if (_degradeDamage > 1) then { _degradeDamage = _degradeDamage * 0.6 };
// Calculate defence distance
_defendDistance = (_speedAdjustment + _defendConstant) / _degradeDamage;
// Calculate time to wait to reach _defendDistance
_wait = _defendDistance / 1000 / _speed * 60 * 60;
```

```
// Set sleep time
_sleep = _time - _wait;



// Only activate once the flight time is longer than the wait time

if (_wait > _sleep) then
{
// Get ammo count
_ammoLeft = _defender getVariable "ammo_left";
_ammoRight = _defender getVariable "ammo_right";

// Get reloading timers
_reloadLeft = _defender getVariable "reloading_left";
_reloadRight = _defender getVariable "reloading_right";

// Check if the vehicle's APS is reloading on default/left side
if ((_block == "left" || _block == "true") && _reloadLeft == 1) then
_canBlock = false;
};

// Check if the vehicle's APS is reloading on right side
if (_block == "right" && _reloadRight == 1) then
{
_canBlock = false;
};
// Check if the vehicles's default/left APS has ammo left
if (_canBlock && (_block == "left" || _block == "true")) then
{
if (_ammoLeft < 1) then
{
_canBlock = false;
};
if (_ammoLeft > 0) then
{
_defender setVariable ["ammo_left", (_ammoLeft-1), true];
// Degrade performance for every use
if (_ammoLeft < 4) then
{
_dDis = (5 / _ammoLeft);
_degradeUses = (_dDis * _dDis) / 1000 / _speed * 60 * 60;
_sleep = _sleep + _degradeUses;
};
};
};
// Check if the vehicle's right APS has ammo left
if (_canBlock &&  _block == "right" ) then
```

```
{
if (_ammoRight < 1) then
{
_canBlock = false;
};
if (_ammoRight > 0) then
{

_defender setVariable ["ammo_right", (_ammoRight-1), true];
// Degrade performance for every use
if (_ammoRight < 4) then
{
_dDis = (5 / _ammoRight);
_degradeUses = (_dDis * _dDis) / 1000 / _speed * 60 * 60;
_sleep = _sleep + _degradeUses;
};
};
};
[_sleep] spawn FNC_RELOAD_DISPLAY; //Added

// Check that threat is not more than 10m from top
_defenderAlt = getPosASL _defender select 2;
_projectileAlt = getPosASL _projectile select 2;
if ((_projectileAlt - _defenderAlt) > 14) then
{
_canBlock = false;
};
// Destroy the threat
if (alive _projectile && _canBlock) then
{
// Wait until threat is in range
sleep _sleep;
// Remove the threat and create FX
deleteVehicle _projectile;
_sfx = createVehicle ["HelicopterExploSmall", position _projectile, [], 0,
"CAN_COLLIDE"]; //Modified
// Do reload and ammo functions
{
[_defender] spawn
{
_defender = _this Select 0;
_defender setVariable ["reloading_left", 1, true];
sleep 7.2;
_defender setVariable ["reloading_left", 0, true];
};
};
if (_block == "right") then
{
```

```
[_defender] spawn
{
_defender = _this Select 0;
_defender setVariable ["reloading_right", 1, true];
sleep 7.2;
_defender setVariable ["reloading_right", 0, true];


};
};
//Update ammo on GUI
call FNC_UPDATE_GUI; //Added
// AI behavior
_defender setBehaviour "DANGER";
if (_defender != player) then
{
[_defender, _unit] spawn { sleep 0.5 ; (_this Select 0) doWatch (_this Select 1); };
};
};
_do = 0;
};
};
sleep 0.01;
};
};
```

Appendix 2
1(4)

Baked_AIS_init.sqf By Bakerman

```
// FILE                    .       Baked_AIS_init.sqf
// AUTHOR                  .       Bakerman, Extended/modified by Arto Alamäki
// LAST EDIT               .       15/09/2013
// Version                 .       0.1.3
// DESCRIPTION             .       APS Initialization for Baked_AIS (Armour Improvement System)
// LICENSE                 .       GO CRAZY - USE IT - ABUSE IT - CREDIT IF YOU WISH

// ************************************************************
// Call with
// call compile preprocessFile "Baked_AIS\Baked_AIS_init.sqf";
// ************************************************************
// Make sure to only run once
if !(isnil "baps_running") exitwith {hintSilent "!WARNING! CANNOT RUN MULTIPLE INSTANCES OF AIS-APS";};
// Run only on server
if (isServer) then
{
// Get config values
baps_startHint = 0; // Show hint on startup?
baps_startDelay = 0; // Delay script for x seconds
baps_enabledFor = 1; // Enable for types? 0 = M2A1, 1 = ALL TANKS, 2 = ALL ARMOR, 3 = ALL NATO ARMOR, 4 = ALL CSAT ARMOR

// Revert to default values if nil
if (isNil "baps_startHint") then { _startHint = 0 };
if (isNil "baps_startDelay") then { _startDelay = 0 };
if (isNil "baps_enabledFor") then { _enabledFor = 0 };
// Broadcast Variables
publicVariable "baps_startHint";
publicVariable "baps_startDelay";
publicVariable "baps_enabledFor";

onPlayerConnected "[] spawn FNC_ADD_ACTIONS";
};

// Run only on clients
if (!isServer) then
{
// Revert to default values if nil
if (isNil "baps_startHint") then { _startHint = 0 };
if (isNil "baps_startDelay") then { _startDelay = 0 };
if (isNil "baps_enabledFor") then { _enabledFor = 0 };
};

// Pause script for x seconds
```

```
sleep baps_startDelay;
```



```
// Call & cache functions
call compile preprocessFile "Baked_AIS_fnc.sqf"; //Baked_AIS\
call compile preprocessFile "aps_activation.sqf";//Added
call compile preprocessFile "update_GUI.sqf"; //Added
call compile preprocessFile "radar.sqf";//Added

// List of weapons/launchers
baps_weapons = [
"missiles_titan",
"launch_RPG32_F",
"launch_Titan_short_F",
"launch_B_Titan_short_F",
"launch_O_Titan_short_F",
"launch_I_Titan_short_F",

"launch_Titan_F",
"launch_B_Titan_F",
"launch_O_Titan_F",
"launch_I_Titan_F",
"launch_NLAW_F",
"missiles_DAGR",
"missiles_ASRAAM",
"missiles_SCALPEL",
"rockets_Skyfire"];

// List of vehicles that defend
baps_defenders = [];
if (baps_enabledFor == 0) then { // FOR M2A1 SLAMMER ONLY
baps_defenders = [
"B_MBT_01_cannon_F"]; // Merkava Mk4
};
if (baps_enabledFor == 1) then { // FOR MBT ONLY
baps_defenders = [
"B_MBT_01_cannon_F", // Merkava Mk4
"O_MBT_02_cannon_F"]; // T-100
};
if (baps_enabledFor == 2) then { // FOR ALL ARMORED VEHICLES
baps_defenders = [
"B_APC_Tracked_01_rcws_F",
"B_APC_Wheeled_01_cannon_F",
"B_APC_Tracked_01_CRV_F",
"B_APC_Tracked_01_AA_F",
"B_MBT_01_cannon_F",
"B_MBT_01_arty_F",
"B_MBT_01_mlrs_F",
"O_MBT_02_cannon_F",
"O_MBT_02_arty_F",
```

```
"O_APC_Tracked_02_AA_F",
"O_APC_Tracked_02_cannon_F",

"O_APC_Wheeled_02_rcws_F",
"I_APC_Wheeled_03_cannon_F"];
};
if (baps_enabledFor == 3) then { // BLUFOR / NATO ONLY
baps_defenders = [
"B_APC_Tracked_01_rcws_F",
"B_APC_Wheeled_01_cannon_F",
"B_APC_Tracked_01_CRV_F",
"B_APC_Tracked_01_AA_F",
"B_MBT_01_cannon_F",
"B_MBT_01_arty_F",
"B_MBT_01_mlrs_F"];
};
if (baps_enabledFor == 4) then { // OPFOR / CSAT ONLY
baps_defenders = [
"O_MBT_02_cannon_F",
"O_MBT_02_arty_F",
"O_APC_Tracked_02_AA_F",
"O_APC_Tracked_02_cannon_F",
"O_APC_Wheeled_02_rcws_F"];
};
// Get an array of every land vehicle and assign ammo count
// Add actions for all units and vehicles
baps_vehiclelist = [];
[] spawn
{
while { true } do
{
{
if (!(_x in baps_vehiclelist) && (alive _x) && ((typeOf _x) in baps_defenders)) then
{
baps_vehiclelist set [count baps_vehiclelist, _x];
_x setVariable ["ammo_left", 4, true];
_x setVariable ["ammo_right", 4, true];
_x setVariable ["reloading_left", 0, true];
_x setVariable ["reloading_right", 0, true];
_x setVariable ["aps_active", 1, true]; //added
_X addAction ["Tank Commander GUI",
"call_GUI.sqf",true,1,false,true,"","commander _target == _this"];//added
};
} forEach vehicles;
[] spawn FNC_ADD_ACTIONS;
sleep 10;
};
};
```

```
baps_unitsActionList = [];
baps_vehiclesActionsList = [];

FNC_ADD_ACTIONS =
{
{
if (!(_x in baps_unitsActionList)) then
{
_x addEventHandler ["Fired",{_this spawn FNC_BAPS_FIRED}];
baps_unitsActionList set [count baps_unitsActionList, _x];
};
} forEach allUnits;
{
if (!(_x in baps_vehiclesActionsList)) then
{
_x addEventHandler ["Fired",{_this spawn FNC_BAPS_FIRED}];

baps_vehiclesActionsList set [count baps_vehiclesActionsList, _x];
};
} forEach vehicles;
};

// Hint that system is active
if (baps_startDelay == 1) then {
hintSilent "AIS-APS SCRIPT ACTIVE";
};
// AIS-APS is active has run
baps_running = true;
```

Appendix 3

```
//Get projectile direction and compare to sectors
FNC_DIRECTION_CHECK=
{
private ["_relativeDir", "_radarDir"];
_relativeDir= _this select 0;

//front
if (_relativeDir > 316 || _relativeDir < 45) then
{
_radarDir = 0;
[_radarDir] spawn FNC_RADAR;
};
//right
if (_relativeDir > 46 && _relativeDir < 135) then
{
_radarDir = 1;
[_radarDir] spawn FNC_RADAR;
};
//rear
if (_relativeDir > 136 && _relativeDir < 225) then
{
_radarDir = 2;
[_radarDir] spawn FNC_RADAR;
};
//left
if (_relativeDir > 226 && _relativeDir < 315) then
{
_radarDir = 3;
[_radarDir] spawn FNC_RADAR;
};
};
//Radar visual effect
FNC_RADAR=
{
private ["_radarDir"];
_radarDir = _this select 0;
_imgArray=["radar_img\radar_front.paa","radar_img\radar_right.paa",
"radar_img\radar_rear.paa", "radar_img\radar_left.paa"]; //Array for radar images
_image = _imgArray select _radarDir; //Select image according to the direction check
_blink = 0;
//Radar blink effect
while {_blink < 9} do
{
ctrlSetText [1200, _image];
sleep 0.2;
ctrlSetText [1200, "radar_img\radar_base.paa"];
_blink = _blink+1;
sleep 0.2;
};
};
```

Appendix 4

```
//Warning sound for the radar
private ["_projectile", "_defender"];
_projectile = _this select 0;
_defender = _this select 1;

//Plays the sound only when the commander seat is filled
if (player == assignedCommander _defender)then
{
while {alive _projectile} do
{
playSound3D ["A3\Sounds_F\sfx\alarm.wss", player, true, getPosASL player,1,1,0];
sleep 0.8;
};
};
```

```
//Dialogs for GUI
class tank_commander_GUI
{
idd=-1;
movingenable=1;

class controls
{
//////////////////////////////////////////////////////
// GUI EDITOR OUTPUT START
//////////////////////////////////////////////////////
class bg_box. BOX
{
icd = -1;
text = "";
x = 0.304134 * safezoneW + safezoneX;
y = 0.244845 * safezoneH + safezoneY;
w = 0.200403 * safezoneW;
h = 0.285949 * safezoneH;
};
class GUI_frame. RscFrame
{
idc = 1800;
moving = 1;
text = "Commander Panel"; //--- ToDo. Localize;
x = 0.304134 * safezoneW + safezoneX;
y = 0.244845 * safezoneH + safezoneY;
w = 0.200403 * safezoneW;
h = 0.285949 * safezoneH;
};
class toggle_APS. RscButton
{
idc = 1600;
text = "APS"; //--- ToDo. Localize;
x = 0.315474 * safezoneW + safezoneX;
y = 0.260241 * safezoneH + safezoneY;
w = 0.0412349 * safezoneW;
h = 0.0439922 * safezoneH;
action = "_this spawn FNC_APS_ACTIVATION";
};
class APS_status. RscText
{
idc = 1000;
text = ""; //--- ToDo. Localize;
x = 0.365987 * safezoneW + safezoneX;
y = 0.264641 * safezoneH + safezoneY;
w = 0.0566979 * safezoneW;
```

```
h = 0.0329942 * safezoneH;
};
class ammo_text. RscText
{
idc = 1001;
text = "AMMO"; //--- ToDo. Localize;
x = 0.324752 * safezoneW + safezoneX;
y = 0.335029 * safezoneH + safezoneY;
w = 0.0309261 * safezoneW;
h = 0.0329942 * safezoneH;
};
class left_text. RscText
{
idc = 1002;
text = "L"; //--- ToDo. Localize;
x = 0.314443 * safezoneW + safezoneX;
y = 0.376821 * safezoneH + safezoneY;
w = 0.0154631 * safezoneW;
h = 0.0329942 * safezoneH;
};
class ammo_right. RscText
{
idc = 1003;
text = "R"; //--- ToDo. Localize;
x = 0.355678 * safezoneW + safezoneX;
y = 0.376821 * safezoneH + safezoneY;
w = 0.0154631 * safezoneW;
h = 0.0329942 * safezoneH;
};
class left_ammo_count. RscText
{
idc = 1004;
text = ""; //--- ToDo. Localize;
x = 0.313412 * safezoneW + safezoneX;
y = 0.416416 * safezoneH + safezoneY;
w = 0.0154631 * safezoneW;
h = 0.0329942 * safezoneH;
};
class right_ammo_count. RscText
{
idc = 1005;
text = ""; //--- ToDo. Localize;
x = 0.354648 * safezoneW + safezoneX;
y = 0.416416 * safezoneH + safezoneY;
w = 0.0154631 * safezoneW;
h = 0.0329942 * safezoneH;
};
class reload_indicator. RscText
```

Appendix 5
3(3)

```
{

idc = 1006;
text = "READY"; //--- ToDo. Localize;
x = 0.31135 * safezoneW + safezoneX;
y = 0.467006 * safezoneH + safezoneY;
w = 0.0566979 * safezoneW;
h = 0.0439922 * safezoneH;
};
class base_picture. RscPicture
{
idc = 1200;
text = "";
x = 0.383512 * safezoneW + safezoneX;
y = 0.317431 * safezoneH + safezoneY;
w = 0.115000 * safezoneW;
h = 0.197965 * safezoneH;
};
//////////////////////////////////////////////////////
// GUI EDITOR OUTPUT END
//////////////////////////////////////////////////////
};
};
```

Appendix 6

```
_vehicle = nearestObject [player, "LandVehicle"];
_status = _vehicle getVariable "aps_active";
handle = createDialog "tank_commander_GUI";

if (_status == 1) then
{
ctrlSetText [1000, "ONLINE"];
ctrlSetText [1200, "radar_img\radar_base.paa"];
}
else
{
ctrlSetText [1000, "OFFLINE"];
ctrlSetText [1200, ""];
};

call FNC_UPDATE_GUI;
```

Appendix 7

```
FNC_APS_ACTIVATION=
{
_vehicle = nearestObject [player, "LandVehicle"];//Nearest vehicle to the player
_status = _vehicle getVariable "aps_active"; //Check if AIS is active on vehicle

//turn APS ON/OFF
if (_status == 1) then
{
_vehicle setVariable ["aps_active", 0, true];
ctrlSetText [1000, "OFFLINE"];//Status message
ctrlSetText [1200, ""];//Hide radar image
}
else
{
_vehicle setVariable ["aps_active", 1, true];
ctrlSetText [1000, "ONLINE"];//status message
ctrlSetText [1200, "radar_img\radar_base.paa"]; //Show radar base image
};
};
```

Appendix 8

```
//Updates ammo count for the GUI
FNC_UPDATE_GUI=
{
_vehicle = nearestObject [player, "LandVehicle"]; //Nearest vehicle to the player
_leftUpdate = _vehicle getVariable "ammo_left"; //Retrieve ammo count
_rightUpdate = _vehicle getVariable "ammo_right";
_doUpdate = 1;

if (_doUpdate == 1) then
{
ctrlSetText [1004, format ["%1", _leftUpdate]];
ctrlSetText [1005, format ["%1", _rightUpdate]];
};
};

//Reloading text and delay for the AIS
FNC_RELOAD_DISPLAY=
{
private ["_sleep"];
_sleep = _this select 0;
_sleep= _sleep + 7.2; //Reloading time in seconds

ctrlSetText [1006, "RELOADING"]; //Set reload text
sleep _sleep;
ctrlSetText [1006, "READY"];
};
```

```
// Control types
#define CT_STATIC           0
#define CT_BUTTON           1
#define CT_EDIT             2
#define CT_SLIDER           3
#define CT_COMBO            4
#define CT_LISTBOX          5
#define CT_TOOLBOX          6
#define CT_CHECKBOXES       7
#define CT_PROGRESS         8
#define CT_HTML             9
#define CT_STATIC_SKEW      10
#define CT_ACTIVETEXT       11
#define CT_TREE             12
#define CT_STRUCTURED_TEXT  13
#define CT_CONTEXT_MENU     14
#define CT_CONTROLS_GROUP   15
#define CT_SHORTCUTBUTTON   16
#define CT_XKEYDESC         40
#define CT_XBUTTON          41
#define CT_XLISTBOX         42
#define CT_XSLIDER          43
#define CT_XCOMBO           44
#define CT_ANIMATED_TEXTURE 45
#define CT_OBJECT           80
#define CT_OBJECT_ZOOM      81
#define CT_OBJECT_CONTAINER 82
#define CT_OBJECT_CONT_ANIM 83
#define CT_LINEBREAK        98
#define CT_USER             99
#define CT_MAP              100
#define CT_MAP_MAIN         101
#define CT_LISTNBOX         102

// Static styles
#define ST_POS            0x0F
#define ST_HPOS           0x03
#define ST_VPOS           0x0C
#define ST_LEFT           0x00
#define ST_RIGHT          0x01
#define ST_CENTER         0x02
#define ST_DOWN           0x04
#define ST_UP             0x08
#define ST_VCENTER        0x0C
#define ST_GROUP_BOX      96
#define ST_GROUP_BOX2     112
```

```
#define ST_ROUNDED_CORNER  ST_GROUP_BOX + ST_CENTER
#define ST_ROUNDED_CORNER2 ST_GROUP_BOX2 + ST_CENTER

#define ST_TYPE          0xF0
#define ST_SINGLE        0x00
#define ST_MULTI         0x10
#define ST_TITLE_BAR     0x20
#define ST_PICTURE       0x30
#define ST_FRAME         0x40
#define ST_BACKGROUND    0x50
#define ST_GROUP_BOX     0x60
#define ST_GROUP_BOX2    0x70
#define ST_HUD_BACKGROUND 0x80
#define ST_TILE_PICTURE  0x90
#define ST_WITH_RECT     0xA0
#define ST_LINE          0xB0
#define ST_SHADOW        0x100
#define ST_NO_RECT       0x200
#define ST_KEEP_ASPECT_RATIO  0x800
#define ST_TITLE         ST_TITLE_BAR + ST_CENTER
// Slider styles
#define SL_DIR           0x400
#define SL_VERT          0
#define SL_HORZ          0x400
#define SL_TEXTURES      0x10

// progress bar
#define ST_VERTICAL      0x01
#define ST_HORIZONTAL    0

// Listbox styles
#define LB_TEXTURES      0x10
#define LB_MULTI         0x20

// Tree styles
#define TR_SHOWROOT      1
#define TR_AUTOCOLLAPSE  2

// MessageBox styles
#define MB_BUTTON_OK     1
#define MB_BUTTON_CANCEL 2
#define MB_BUTTON_USER   4

////////////////
//Base Classes//
////////////////

class RscText
```

```
{
   access = 0;
   idc = -1;
   type = CT_STATIC;
   style = ST_CENTER;
   linespacing = 1;
   colorBackground[] = {0,0,0,0};
   colorText[] = {0.3,0.6,0.4,.8};
   text = "";
   shadow = 2;
   font = "EtelkaNarrowMediumPro";
   SizeEx = 0.04000;
   fixedWidth = 0;
   x = 0;
   y = 0;
   h = 0;
   w = 0;

};

class RscPicture
{
   access = 0;
   idc = -1;
   type = CT_STATIC;
   style = ST_PICTURE;
   colorBackground[] = {0,0,0,0};
   colorText[] = {1,1,1,1};
   font = "Bitstream";
   sizeEx = 0;
   lineSpacing = 0;
   text = "";
   fixedWidth = 0;
   shadow = 0;
   x = 0;
   y = 0;
   w = 0.2;
   h = 0.15;
};

class RscButton
{
   access = 0;
   idc = -1;
   type = CT_BUTTON;
   text = "";
   colorText[] = {0.5,0.3,0.9,.9};
   colorDisabled[] = {0.6,0.1,0.3,0};
```

```
    colorBackground[] = {0.35,0.55,0.15,0.9};
    colorBackgroundDisabled[] = {0,0.0,0};
    colorBackgroundActive[] = {0.15,0.35,0.55,0.7};
    colorFocused[] = {0.75,0.75,0.75,.5};
    colorShadow[] = {0.023529,0,0.0313725,1};
    colorBorder[] = {0.023529,0,0.0313725,1};
    soundEnter[] = {"\ca\ui\data\sound\onover",0.09,1};
    soundPush[] = {"\ca\ui\data\sound\new1",0,0};
    soundClick[] = {"\ca\ui\data\sound\onclick",0.07,1};
    soundEscape[] = {"\ca\ui\data\sound\onescape",0.09,1};
    style = 2;
    x = 0;
    y = 0;
    w = 0.055589;
    h = 0.010000;
    shadow = 2;
    font = "Bitstream";
    sizeEx = 0.02921;
    offsetX = 0.003;
    offsetY = 0.003;
    offsetPressedX = 0.002;
    offsetPressedY = 0.002;
    borderSize = 0;
};
class RscFrame
{
    type = CT_STATIC;
    idc = -1;
    style = ST_FRAME;
    shadow = 2;
    colorBackground[] = {0.5,0.3,0.5,1};
    colorText[] = {1,1,1,0.9};
    font = "Bitstream";
    sizeEx = 0.03;
    text = "";
};
class BOX
{
    type = CT_STATIC;
    idc = -1;
    style = ST_CENTER;
    shadow = 2;
    colorText[] = {1,1,1,1};
    font = "Bitstream";
    sizeEx = 0.02;
    colorBackground[] = { 0.5,0.5,0.5,0.6};
    text = "";
};
```